CS 267

Tricks with Trees

James Demmel

www.cs.berkeley.edu/~demmel/cs267_Spr15

Outline

- A log n lower bound to compute any function in parallel
- Reduction and broadcast in O(log n) time
- Parallel prefix (scan) in O(log n) time
- Adding two n-bit integers in O(log n) time
- Multiplying n-by-n matrices in O(log n) time
- Inverting n-by-n triangular matrices in O($\log^2$ n) time
- Inverting n-by-n dense matrices in O($\log^2$ n) time
- Evaluating arbitrary expressions in O(log n) time
- Evaluating recurrences in O(log n) time

Outline

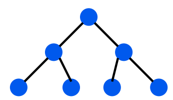
- A log n lower bound to compute any function in parallel
- Reduction and broadcast in O(log n) time
- Parallel prefix (scan) in O(log n) time
- Adding two n-bit integers in O(log n) time
- Multiplying n-by-n matrices in O(log n) time
- Inverting n-by-n triangular matrices in O($\log^2$ n) time
- Inverting n-by-n dense matrices in O($\log^2$ n) time
- Evaluating arbitrary expressions in O(log n) time
- Evaluating recurrences in O(log n) time
- "2D parallel prefix", for image segmentation (Bryan Catanzaro, Kurt Keutzer)
- Sparse-Matrix-Vector-Multiply (SpMV) using Segmented Scan
- Parallel page layout in a browser (Leo Meyerovich, Ras Bodik)

Outline

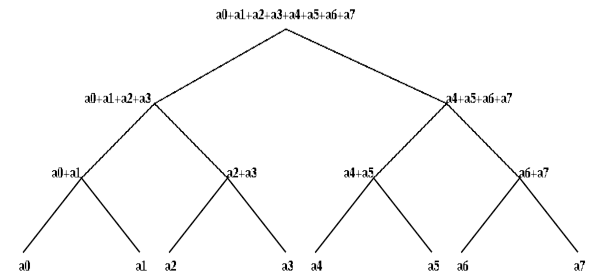
- A log n lower bound to compute any function in parallel
- Reduction and broadcast in O(log n) time
- Parallel prefix (scan) in O(log n) time
- Adding two n-bit integers in O(log n) time
- Multiplying n-by-n matrices in O(log n) time
- Inverting n-by-n triangular matrices in O($\log^2$ n) time
- Inverting n-by-n dense matrices in O($\log^2$ n) time
- Evaluating arbitrary expressions in O(log n) time
- Evaluating recurrences in O(log n) time
- "2D parallel prefix", for image segmentation (Bryan Catanzaro, Kurt Keutzer)
- Sparse-Matrix-Vector-Multiply (SpMV) using Segmented Scan
- Parallel page layout in a browser (Leo Meyerovich, Ras Bodik)
- Solving n-by-n tridiagonal matrices in O(log n) time
- Traversing linked lists
- Computing minimal spanning trees
- Computing convex hulls of point sets…

A log n lower bound to compute any function of n variables

- Assume we can only use binary operations, one per time unit
- After 1 time unit, an output can only depend on two inputs
- Use induction to show that after k time units, an output can only depend on 2^k inputs
  - After $\log_2 n$ time units, output depends on at most n inputs
- A binary tree performs such a computation



Broadcasts and Reductions on Trees

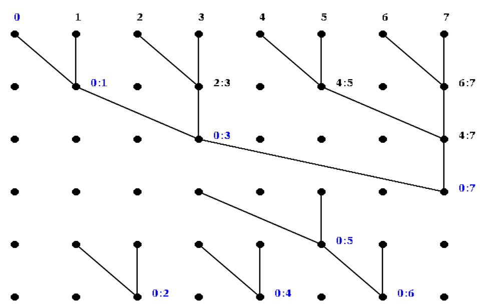


Parallel Prefix, or Scan

- If "+" is an associative operator, and x[0],…,x[p-1] are input data then parallel prefix operation computes

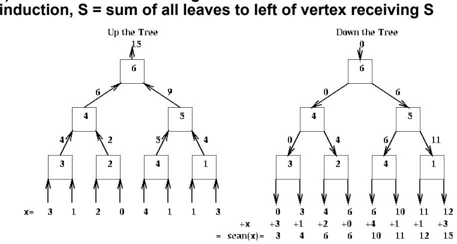
  y[j] = x[0] + x[1] + … + x[j]   for j=0,1,…,p-1

- Notation: j:k means x[j]+x[j+1]+…+x[k], blue is final value



Mapping Parallel Prefix onto a Tree - Details

- Up-the-tree phase (from leaves to root)
  1) Get values L and R from left and right children
  2) Save L in a local register Lsave
  3) Pass sum L+R to parent
- By induction, Lsave = sum of all leaves in left subtree
- Down the tree phase (from root to leaves)
  1) Get value S from parent (the root gets 0)
  2) Send S to the left child
  3) Send S + Lsave to the right child
- By induction, S = sum of all leaves to left of vertex receiving S



E.g., Fibonacci via Matrix Multiply Prefix

$$\boxed{F_{n+1} = F_n + F_{n-1}}$$

$$\boxed{\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}}$$

Can compute all F_n by matmul_prefix on

$$\left[\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right]$$

then select the upper left entry

Slide source: Alan Edelman, MIT

Adding two n-bit integers in O(log n) time

- **Let a = a[n-1]a[n-2]…a[0] and b = b[n-1]b[n-2]…b[0] be two n-bit binary numbers**
- **We want their sum s = a+b = s[n]s[n-1]…s[0]**

```
c[-1] = 0          … rightmost carry bit
for i = 0 to n-1
    c[i] = ( (a[i] xor b[i])  and  c[i-1] )  or  ( a[i]  and  b[i] )   ... next carry bit
    s[i] = ( a[i] xor b[i] ) xor c[i-1]
```

- **Challenge: compute all c[i] in O(log n) time via parallel prefix**

```
for all (0 <= i <= n-1)  p[i] = a[i] xor b[i]      … propagate bit
for all (0 <= i <= n-1)  g[i] = a[i] and b[i]      … generate bit
```

$$\begin{bmatrix} c[i] \\ 1 \end{bmatrix} = \begin{bmatrix} (\ p[i] \text{ and } c[i-1]\ ) \text{ or } g[i] \\ 1 \end{bmatrix} = \begin{bmatrix} p[i] & g[i] \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} c[i-1] \\ 1 \end{bmatrix} = C[i] * \begin{bmatrix} c[i-1] \\ 1 \end{bmatrix}$$

… 2-by-2 Boolean matrix multiplication (associative)

$$= C[i] * C[i-1] * \ldots C[0] * \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

… evaluate each P[i] = C[i] * C[i-1] * … * C[0] by parallel prefix

- **Used in all computers to implement addition - Carry look-ahead**

Other applications of scan = parallel prefix

- **There are many applications of scans, some more obvious than others**
  - add multi-precision numbers (represented as array of numbers)
  - evaluate recurrences, expressions
  - solve tridiagonal systems (but numerically unstable!)
  - implement bucket sort and radix sort
  - to dynamically allocate processors
  - to search for regular expression (e.g., grep)
  - many others…
- **Names: +\ (APL), cumsum (Matlab), MPI_SCAN**
- **Note: 2n operations used when only n-1 needed**

Multiplying n-by-n matrices in O(log n) time

- **For all (1 <= i,j,k <= n) P(i,j,k) = A(i,k) * B(k,j)**
  - cost = 1 time unit, using n^3 processors
- **For all (1 <= i,j <= n) $C(i,j) = \sum_{k=1}^{n} P(i,j,k)$**
  - cost = O(log n) time, using n^2 trees with n^3 / 2 processors

Inverting triangular n-by-n matrices in $O(\log^2 n)$ time

- **Fact:** $\begin{bmatrix} A & 0 \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -B^{-1}CA^{-1} & B^{-1} \end{bmatrix}$

- **Function Tri_Inv(T)** … assume $n = \dim(T) = 2^m$ for simplicity

```
If T is 1-by-1
   return 1/T
else
   … Write T = [A 0]
               [C B]
   In parallel do {
        invA = Tri_Inv(A)
        invB = Tri_Inv(B) }     … implicitly uses a tree
   newC = -invB * C * invA
   Return [invA   0  ]
          [newC  invB]
```

- **time(Tri_Inv(n)) = time(Tri_Inv(n/2)) + O(log(n))**
  - Change variable to m = log n to get time(Tri_Inv(n)) = $O(\log^2 n)$

Inverting Dense n-by-n matrices in $O(\log^2 n)$ time

- **Lemma 1: Cayley-Hamilton Theorem**
  - expression for A^{-1} via characteristic polynomial in A
- **Lemma 2: Newton's Identities**
  - Triangular system of equations for coefficients of characteristic polynomial, where matrix entries = s_k
- **Lemma 3:** $s_k = \mathrm{trace}(A^k) = \sum_{i=1}^{n} A^k[i,i]$
- **Csanky's Algorithm (1976)**

  1) Compute the powers $A^2, A^3, \ldots, A^{n-1}$ by parallel prefix
     cost = $O(\log^2 n)$
  2) Compute the traces $s_k = \mathrm{trace}(A^k)$
     cost = $O(\log n)$
  3) Solve Newton identities for coefficients of characteristic polynomial
     cost = $O(\log^2 n)$
  4) Evaluate A^{-1} using Cayley-Hamilton Theorem
     cost = $O(\log n)$

- **Completely numerically unstable**

Evaluating arbitrary expressions

- **Let E be an arbitrary expression formed from +, -, *, /, parentheses, and n variables, where each appearance of each variable is counted separately**
- **Can think of E as arbitrary expression tree with n leaves (the variables) and internal nodes labeled by +, -, * and /**
- **Theorem (Brent): E can be evaluated in O(log n) time, if we reorganize it using laws of commutativity, associativity and distributivity**
- **Sketch of (modern) proof: evaluate expression tree E greedily by repeatedly**
  - collapsing all leaves into their parents at each time step
  - evaluating all "chains" in E with parallel prefix

Evaluating recurrences

- **Let $x_i = f_i(x_{i-1})$, f_i a rational function, x_0 given**
- **How fast can we compute x_n?**
- **Theorem (Kung): Suppose degree(f_i) = d for all i**
  - If d=1, x_n can be evaluated in O(log n) using parallel prefix
  - If d>1, evaluating x_n takes $\Omega(n)$ time, i.e. no speedup is possible
- **Sketch of proof when d=1**

  $x_i = f_i(x_{i-1}) = (a_i * x_{i-1} + b_i)/(c_i * x_{i-1} + d_i)$ can be written as
  $x_i = num_i / den_i = (a_i * num_{i-1} + b_i * den_{i-1})/(c_i * num_{i-1} + d_i * den_{i-1})$ or

  $$\begin{pmatrix} num_i \\ dem_i \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} * \begin{pmatrix} num_{i-1} \\ den_{i-1} \end{pmatrix} = M_i * \begin{pmatrix} num_{i-1} \\ den_{i-1} \end{pmatrix} = M_i * M_{i-1} * \ldots * M_1 * \begin{pmatrix} num_0 \\ den_0 \end{pmatrix}$$

  Can use parallel prefix with 2-by-2 matrix multiplication
- **Sketch of proof when d>1**
  - degree(x_i) as a function of x_0 is d^i
  - After i parallel steps, degree(anything) $\le 2^i$
  - Computing x_i take $\Omega(i)$ steps

Image Segmentation (1/4)

- Contours are subjective – they depend on perspective
  - Surprise: Humans agree (somewhat)
- Goal: generate contours automatically
  - Use them to break images into separate segments (subimages)
  - J. Malik's group has leading algorithm
  - Enable automatic image search and retrieval ("Find all the pictures with Fred")

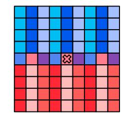
Image

Human Generated Contours

Machine Generated Contours

Image Segmentation (2/4)

- Think of image as matrix A(i,j) of pixels
  - Each pixel has separate R(ed), G(reen), B(lue) intensities
- Bottleneck (so far) of Malik's algorithm is to compute other matrices indicating whether pixel (i,j) likely to be on contour
  - Ex: C(i,j) = average "R intensity" of pixels in rectangle above (i,j) – average "R intensity" of pixels in rectangle below (i,j)
  - C(i,j) large for pixel (i,j) marked with ⊗ , so (i,j) likely to be on contour

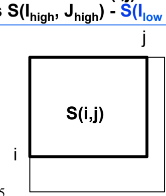


- Algorithm eventually computes eigenvectors of sparse matrix with entries computed from matrices like C
  - Analogous to graph partitioning in later lecture

Image Segmentation (3/4)

- Bottleneck: Given A(i,j), compute C(i,j) where
  - Sa(i,j) = sum of A(i,j) for entries in k x (2k+1) rectangle above A(i,j)

    $= \Sigma$ A(r,s) for $i-k \le r \le i-1$ and $j-k \le s \le j+k$
  - Sb(i,j) = similar sum of rectangle below A(i,j)
  - C(i,j) = Sa(i,j) – Sb(i,j)
- Approach (Bryan Catanzaro)
  - Compute S(i,j) = Σ A(r,s) for $r \le i$ and $s \le j$
  - Then sum of A(i,j) over any rectangle ($I_{low} \le i \le I_{high}$, $J_{low} \le j \le J_{high}$) is $S(I_{high}, J_{high}) - S(I_{low}-1, J_{high}) - S(I_{high}, J_{low}-1) + S(I_{low}-1, J_{low}-1)$



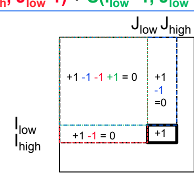
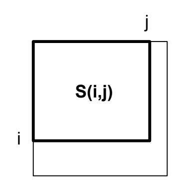


Image Segmentation (4/4)

- New Bottleneck: Given A(i,j), compute S(i,j) where
  - S(i,j) = Σ A(r,s) for $r \le i$ and $s \le j$
- "2 dimensional parallel prefix"
  - Do parallel prefix independently on each row of A(i,j) :
    - $S_{row}(i,j) = \Sigma$ A(i,s) for $s \le j$
  - Do parallel prefix independently on each column of S_{row}
    - S(i,j) = Σ $S_{row}(r,j)$ for $r \le i$ = Σ A(r,s) for $s \le j$ and $r \le i$



Sparse-Matrix-Vector-Multiply (SpMV) y = A*x Using Segmented Scan (SegScan)

° Segscan computes prefix sums of arbitrary segments

Segscan ([3, 1, 4, 5, 6, 1, 2, 3],
[T, F, F, T, T, F, F, T])
= [3, 4, 8, 5, 6, 7, 9, 3]

° Use CSR format of Sparse Matrix A, store x densely

$$A = \begin{bmatrix} 1 & 0 & 2 & 3 & 0 \\ 2 & 4 & 0 & 0 & 5 \\ 3 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad x = \begin{bmatrix} 7 \\ 8 \\ 2 \\ 1 \\ 3 \end{bmatrix}$$

Val = [1 2 3 2 4 5 3 1]
Col_Ind = [1 3 4 1 2 5 1 5]
Row_Ptr=[1 4 7 9]

° Create array P of all nonzero A(i,j)*x(j) = Val(k)*x(Col_Ind(k))

P = [7 4 3 14 32 15 21 3]

° Create array S showing where segments (rows) start

S = [T F F T F F T F]

° Compute SegScan(P, S) =

[7 11 14 14 46 61 21 24]

° Extract A*x = [14 61 24]

° www.cs.cmu.edu/afs/cs.cmu.edu/project/scandal/public/papers/CMU-CS-93-173.ps.Z

Page layout in a browser

° Applying layout rules to html description of a webpage is a bottleneck, scan can help

° Simplest example

- Given widths [x_1, x_2, … , x_n] of items to display on page, where should each item go?
- Item j starts at $x_1 + x_2 + \ldots + x_{j-1}$

° Real examples have complicated constraints

- Defined by general trees, since in html each object to display can be composed of other objects
- To get location of each object, need to do preorder traversal of tree, "adding up" constraints of previous objects
- Scan can do preorder traversal of any tree in parallel
  - Not just binary trees

° Ras Bodik, Leo Meyerovich

Summary of tree algorithms

° Lots of problems can be done quickly - in theory - using trees

° Some algorithms are widely used

- broadcasts, reductions, parallel prefix
- carry look ahead addition

° Some are of theoretical interest only

- Csanky's method for matrix inversion
- Solving tridiagonal linear systems (without pivoting)
- Both numerically unstable
- Csanky needs too many processors

° Embedded in various systems

- MPI, Split-C, Titanium, NESL, other languages
- CM-5 hardware control network